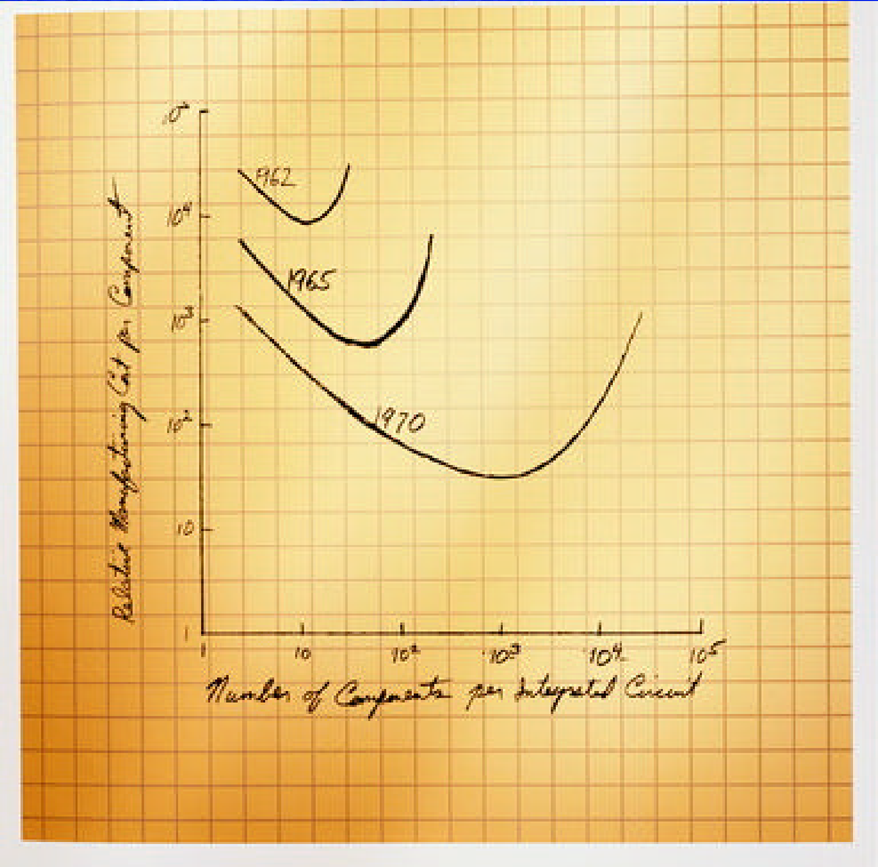
Validating A Modern Microprocessor

Bob Bentley
Intel Corporation
Enterprise Microprocessor Group
Hillsboro, Oregon U.S.A.

Moore's Law - 1965



Source: Intel Museum

Moore's Law - 40 Years Later

Process Name	P854	P856	P858	Px60	P1262	P1264	P1266
1st Production	1995	1997	1999	2001	2003	2005	2007
Lithography	0.35μm	0.25μm	0.18μm	0.13μm	90nm	65nm	45nm
Gate Length	0.35μm	0.20μm	0.13μm	<70nm	<50nm	<35nm	<25nm
Wafer Size (mm)	200	200	200	200/300	300	300	300

A new process every two years

Source: Intel

300mm Semiconductor Economics

Fab	$3 billion
Pilot line	$1-2 billion
R&D process team	$0.5-1 billion

$5 billion investment requires high volume to achieve reasonable unit cost

intel

Source: Intel

The Validation Challenge

- **Microprocessor validation continues to be driven by the economics of Moore's Law**
  - **Each new process generation doubles the number of transistors available to microprocessor architects and designers**
  - **Some of this increase is consumed by larger structures (caches, TLB, etc.), which have no significant impact to validation**
  - **The rest goes to increased complexity:**
    - **Out-of-order, speculative execution machines**
    - **Deeper pipelines**
    - **New technologies (Hyper-Threading, 64-bit extensions, virtualization, security, …**
    - **Multi-core designs**
  - **Increased complexity => increased validation effort and risk**

High volumes magnify the cost of a validation escape

Microprocessor Design

Microprocessor Design Scope

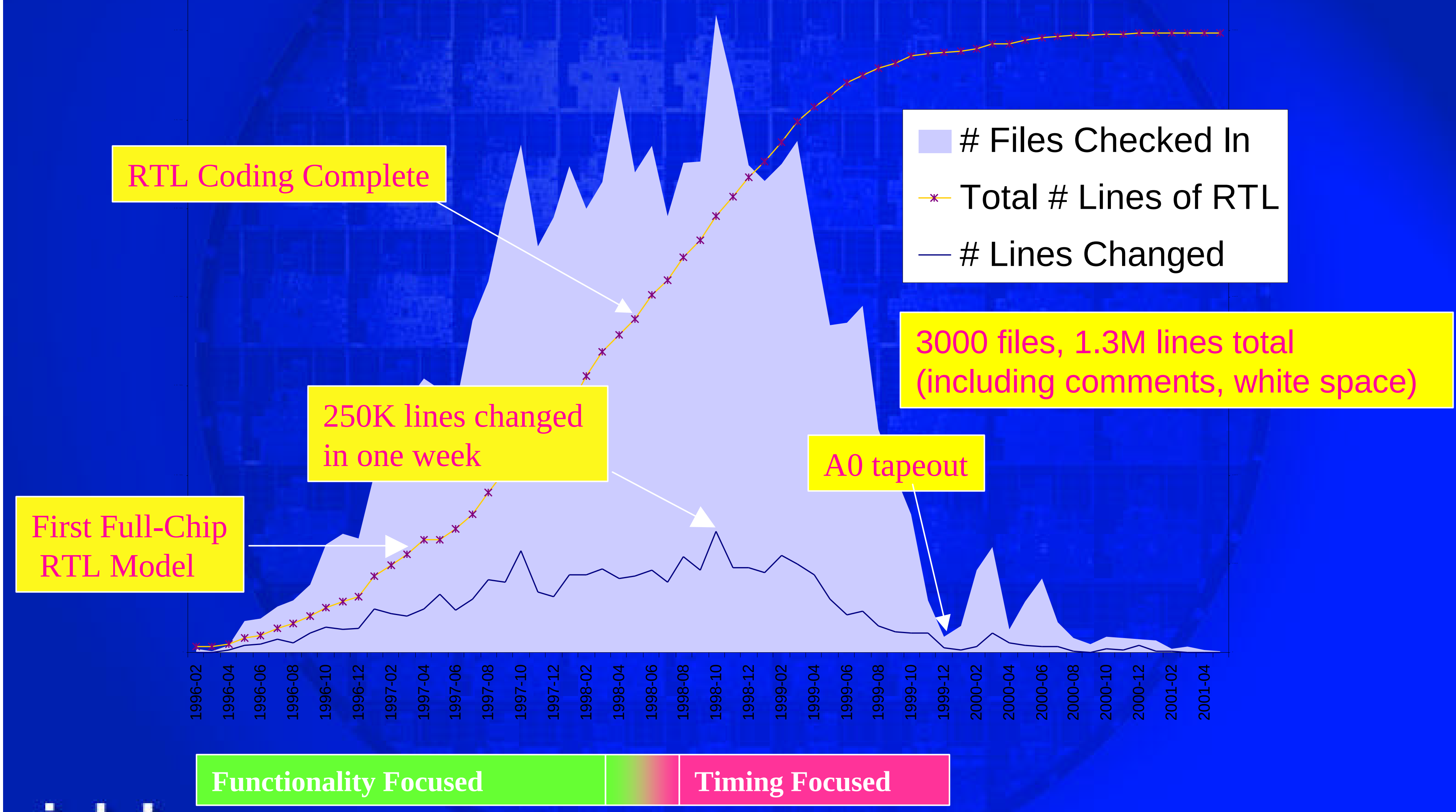
- Typical lead CPU design requires:
  - 500+ person design team:
    - logic and circuit design
    - physical design
    - validation and verification
    - design automation
  - 2-2½ years from start of RTL development to A0 tapeout
  - 9-12 months from A0 tapeout to production qual (may take longer for workstation/server products)

One design cycle = 2 process generations

Pentium® 4 Processor

- RTL coding started: 2H'96
  - First cluster models released: late '96
  - First full-chip model released: Q1'97
- RTL coding complete: Q2'98
  - "All bugs coded for the first time!"
- RTL under full ECO control: Q2'99
- RTL frozen: Q3'99
- A-0 tapeout: December '99
- First packaged parts available: January 2000
- First samples shipped to customers: Q1'00
- Production ship qualification granted: October 2000

RTL – A Moving Target



Microprocessor Validation

RTL validation environment

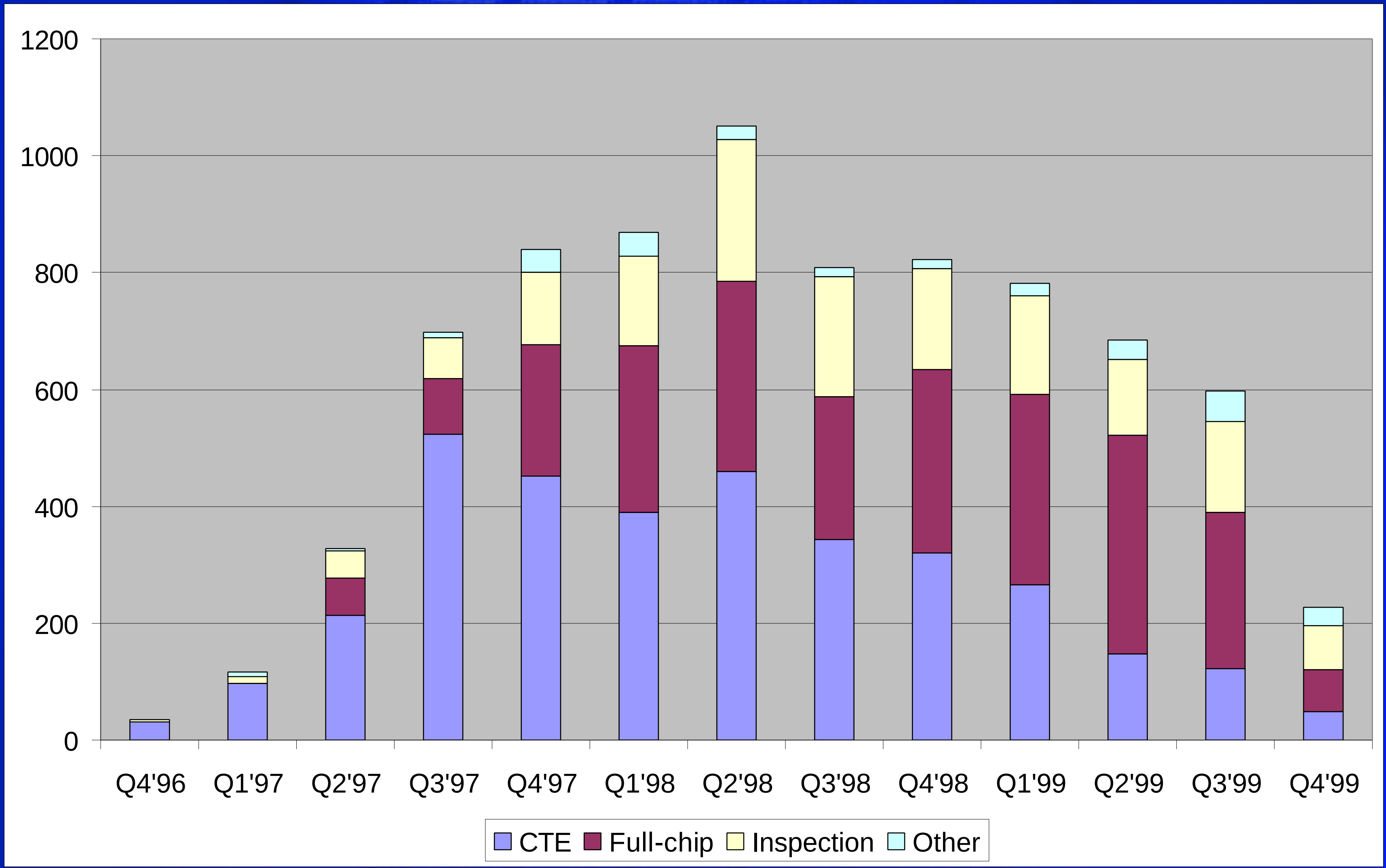
- **RTL model is MUCH slower than real silicon**
  - **A full-chip simulation with checkers runs at ~20 Hz on a Pentium® 4 class machine**
  - **We use a compute farm containing ~6K CPUs running 24/7 to get tens of billions of simulation cycles per week**
  - **The sum total of Pentium® 4 RTL simulation cycles run prior to A0 tapeout < *1 minute on a single 2 GHz system***
- **Pre-silicon validation has some advantages …**
  - **Fine-grained (cycle-by-cycle) checking**
  - **Complete visibility of internal state**
  - **APIs to allow event injection**
- **… but no amount of dynamic validation is enough**
  - **A single dyadic extended-precision (80-bit) FP instruction has O(10**50) possible combinations**
  - **Exhaustive testing is impossible, even on real silicon**

intel

Pentium® 4 Formal Verification

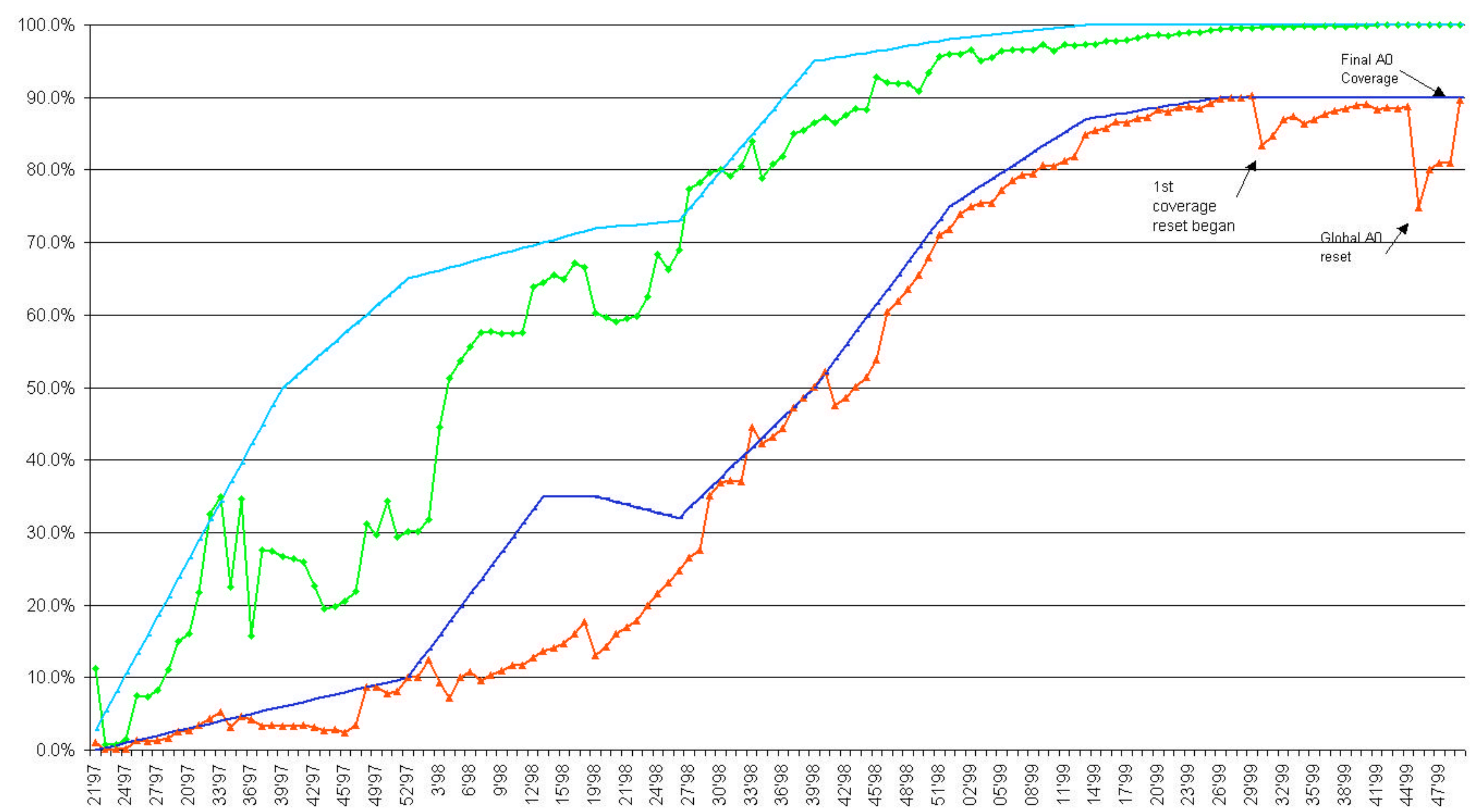
- First large-scale effort at Intel (~60 person years) to apply formal verification techniques to CPU design
  - Applying FV to a moving target is a big challenge!
- Mostly model checking, with some later work using theorem proving to connect FP proofs to IEEE 754
- More than 14,000 properties in key areas:
  - FP Execution units
  - Instruction decode
  - Out-of-order control mechanisms
- Found ~20 "high quality" bugs that would have been hard to detect by dynamic testing
- No silicon bugs found to date in areas proved by FV ☺

Sources of Bugs

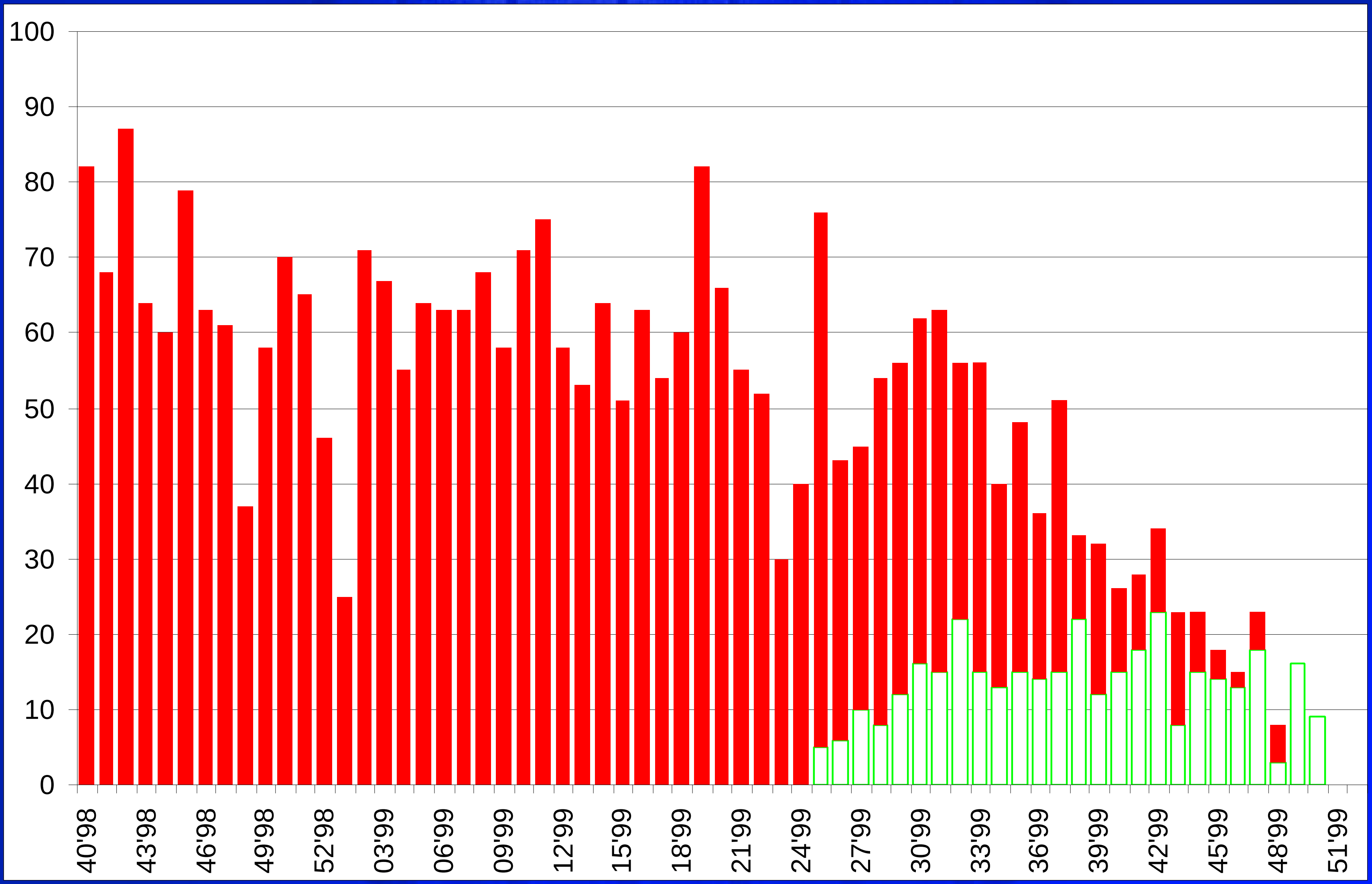


intel

Unit-Level Coverage



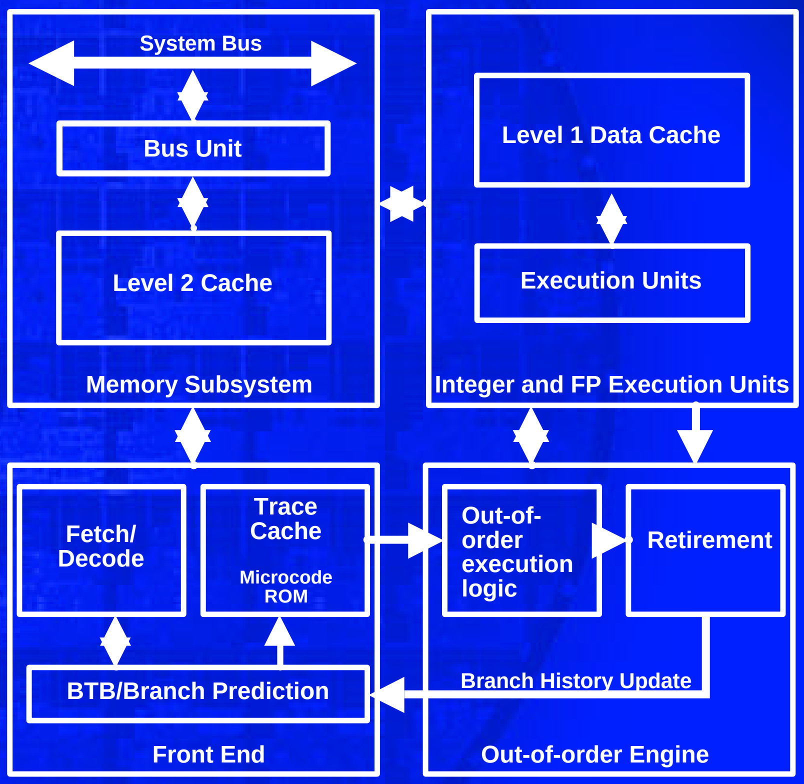
Pre-silicon Bug Rate



Formal Verification

Pentium® 4 Formal Property Verification

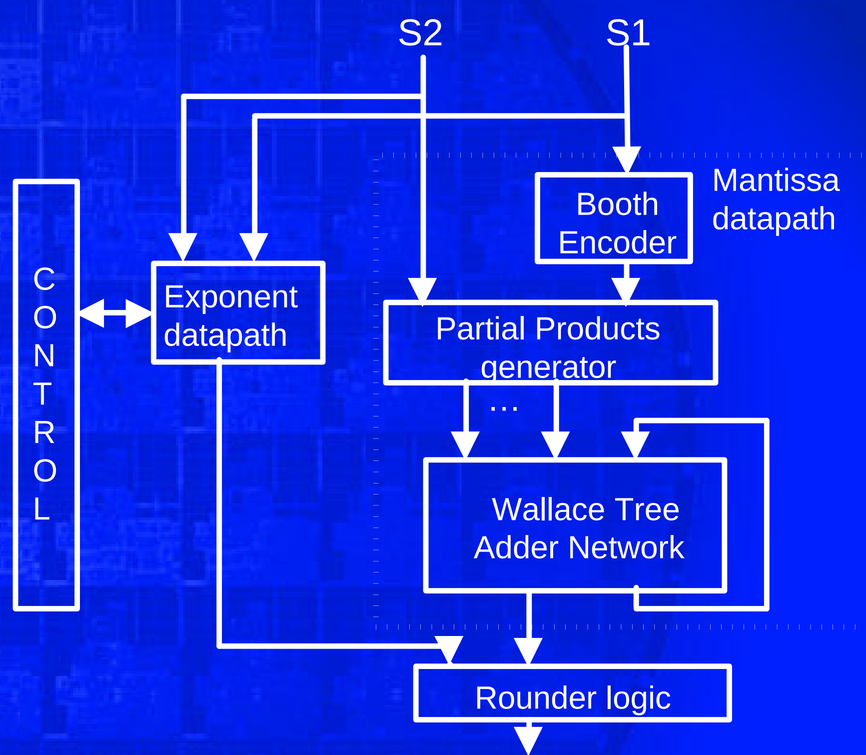
- **Objective:**
  - Complement other validation activities
  - Correctness, not bug hunting
- **Strategy:**
  - Prove unit properties first, then multiple-unit protocols & assumptions
  - Maintain properties in face of an evolving design
- **Tools:**
  - Model checking
  - Symbolic simulation
  - Theorem proving



Pentium® 4 Basic Block Diagram

Floating Point Verification - Multiplication

- Verified adherence to IEEE 754 spec including results, flags & faults
- Design specified as very low-level RTL
- Huge capacity challenge for traditional MC techniques
- Verification required multiple person years and iterations



- Ultimate solution: combined bit-vector level checked results using theorem proving
- Proof framework used on subsequent designs

Combining Formal and Dynamic Verification

- On Pentium® 4, we treated FPV and dynamic verification as essentially independent activities
- Current projects are seeking to exploit synergy between the two techniques
- Currently using SAT solver technology as a bridge between the two worlds
  - SAT solvers provide much greater capacity, reducing or eliminating the need for problem decomposition
  - Allows us to do bug hunting (falsification) in addition to verification
  - Use dynamic validation to confirm or refute counter-examples

High-Level Formal Verification

- ***Formal Design Process:*** **Work with architects & designers while design is being developed**
  - **Abstract model of architecture built and formally checked**
    - **Fast focus on algorithm problems**
    - **Intensive and quick feedback for every definition change**
    - **Find errors in early design approaches**
- **Goal: Completely verify architectural correctness**
  - **Avoid finding architectural bugs late in design cycle, when fixing them is much more difficult and costly**
  - **Drive rigor, completeness and correctness of the high level design**
    - **Speed up design phase based upon complete specification**
    - **Cleaner, simpler design – fewer warts**

Verifying Architectural Correctness

- Top Level Specification
- Architectural Feature Model (AFM)
  - Declare players, describe their roles in upholding top-level specification
  - Model check for violations
- Microarchitectural Block Model (UBM)
  - Player event oriented detail: case-by-case description of how each block reacts to communications
  - Verify by inserting UBM into AFM
  - Fully expose and understand uarch and role in supporting top level
  - Analyze uarch alternatives: UBMs can be developed fairly quickly
- Microarchitectural State Model (USM)
  - One-to-one correspondence with final set of UBMs
  - Describe uarch in a state-centric way

Future Opportunities & Challenges

Tools & Methodology

- **Full verification with SAT-based model checkers**
  - Currently only feasible on simple examples
    - Overlapping loops/feedback in real designs cannot be handled
  - Explicit induction schemes
- **Accumulate design coverage for SAT-based techniques**
  - Need to integrate formal and dynamic (simulation) coverage to fully understand coverage holes
- **Expand formal methods beyond functional correctness**
  - Other areas of concern include performance, power, timing …
  - These areas may be amenable to formal analysis
- **Other tool opportunities:**
  - Parallel and Distributed Checkers
  - Debugging, interactive Model Checking

Methodology drivers

- Regression
  - RTL is “live”, and changes frequently until the very last stages of the project
  - Model checking automation at lower levels allows regression to be automated and provides robustness in the face of ECOs
- Debugging
  - Need to be able to demonstrate FV counter-examples to designers and architects
  - Designers want a dynamic test that they can simulate
  - Waveform viewers, schematic browsers, etc. can help to bridge the gap
- Verification in the large
  - Proof design: how do we approach the problem in a systematic fashion?
  - Proof engineering: how do we write maintainable and modifiable proofs?

Reasoning

- Integrated reasoning within checker
  - Current status:
    - Theorem provers have integrated decision procedures, model checkers don't have reasoning capability
    - Not much improvement in exhaustive MC capacity, need mechanical assistance for problem decomposition
  - Want lightweight framework to direct MC/BMC proof strategy
    - Case split, abstraction, symmetry detection, "what if", …
  - User guided problem decomposition
    - Standard HDLs make it difficult for FV to automatically identify symmetry
- Reasoning about changes
  - Want to fully verify design during development, but real designs go through many iterations

intel

Other Challenges

- **Dealing with constantly-changing specifications**
  - Specification changes are a reality in design
  - Properties and proofs should be readily adapted
  - How to engineer agile and robust regressions?
- **Protocol Verification**
  - This problem has always been hard
  - Getting harder (more MP) and more important (intra-die protocols make it more expensive to fix bugs)
- **Verification of embedded software**
  - S/W for large SoCs has impact beyond functional correctness (power, performance, …)
  - Not all S/W verification techniques apply because H/W abstraction is less feasible
  - One example is microcode verification